FIG. 12 is a flow chart of initialization of the H_code, H_offset and H_seed tables. The starting seed (i.e. the first code word with the smallest numeric value for a given number of bits) is initialized to zero. The number of bits (i.e. the code word length) in the code word is initialized to zero.

Then, as long as i is less than 16, i is incremented by 1, the seed is doubled and stored in H_seed(i). The code is set to seed and then seed is incremented by a number of i-bit codes and j is initialized to zero.

As long as j is less than the number of codes L_i, R/S is set to the next $V_{i,j}$ symbol value in the Define Huffman Table (DHT) marker segment. If the number of bits in the code is less than six, the number of bits and the code are packed into H_code(RS) with three most significant bits in the byte set to the number of bits in the code and the low order five bits set to the code. If i is greater than five, the H_code(RS) is set to the number of bits. Then H_offset(RS) is set to j, since this is the j+1th code of the i-bit codes. Both j and code are incremented for both paths.

For software implementations, the H_seed table entries are likely to be implemented as integers of at least sixteen bits since the longest seed is sixteen bits. For hardware, the nth entry only needs to be n bits because it is the seed for codes of n bits. For simplicity in FIG. 12, the H_seed table is initialized for seventeen entries (0–16). The flow chart shown in FIG. 13 can only use the 6th through 16th entries Again, hardware embodiments may take advantage of this to reduce the number of loadable registers.

Each DC and AC Huffman table needs a set of H_seed, H_code and H_offset tables. Fortunately, for baseline implementations, the DC H_code and H_offset tables need only twelve entries since R=0 and the maximum value of S is 11. For generic implementations, the maximum value of S is 16, making a maximum of seventeen entries for these tables.

For simplicity, the H_code and H_offset tables are likely to have 256 entries each because most of the R/S bytes may be used. The entries in these tables are, at most, a byte. Therefore, instead of needing a 16-bit code word entries plus more bits to signify the number of bits in the code words, a small H_seed table is used for the longer code words and only an extra byte is needed to calculate the final code word of that size. All of the short codes (i.e. less than five bits) are contained in the H_code byte. Since the more frequent symbols will have shorter codes, a single byte access to storage contains everything needed for the most commonly coded symbols.

FIG. 13 provides additional detail for the encode R/S block **1003** of FIG. 11. A temporary variable or register tem is set to the contents of the H_code(RS). The value of n is set to the three most significant bits of the byte. If n is zero, then n is set to tem and the code is generated by adding H_seed(n) to H_offset(RS). Otherwise, code is set to the low five bits of tem. The compressed data is shifted left n bits and the code put into those n low order bits. Comp is treated like a continuous barrel shifter and the process of outputting 32-bit words is well-known to those skilled in the art.

In the prior art, n and the bit pattern of up to 16 bits are stored for up to 256 R/S symbols. In accordance with the invention, however n and the code word are stored in a byte when both can be stored in a total of eight bits or less. This allows the Huffman tables to be considerably shortened and is particularly advantageous since most of the codes encountered will be short enough for this format to be used. Use of this format is indicated by a non-zero value in the three MSB locations of the byte.

For all longer bit patterns, only a length indication is stored as S and the high order nibble is set to 0000 to indicate a different format. For these longer codes, the length is used as an index to the seed (i.e. the first code value of a particular bit length). A corresponding offset (which fits in a byte since the maximum offset is 255) can then be determined from a look-up table to add to the seed to create the code value. After the R/S byte is encoded and stored, the extra S bits are encoded, preferably by shifting to develop a variable length from the two bytes of each AC coefficient value in the intermediate data format of FIG. 10.

In view of the foregoing, it is seen that the invention provides increased efficiency of data processing for encoding, and greatly increased efficiency of computation and storage of tables for indexing into the Huffman tables. These advantages are produced in a manner which is fully compliant with the JPEG standard and are applicable to all applications of Huffman coding.

While the invention has been described in terms of a single preferred embodiment in terms of the JPEG standard. those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims. For example, these fast encoding and decoding methods apply to any Huffman table which is constructed from the number of codes of each length in ascending order. Further, those skilled in the art will be enabled from the above description of the invention to apply the above-described techniques to Huffman tables constructed in descending order.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is as follows:

1. A method of Huffman encoding symbols comprising steps of

defining a seed value for the first occurrence of a code of a given length in a table,

storing a length of a code word,

storing said length and said code word in a first format when a number of bits of said number and said code word are less than or equal to a predetermined number of bits, and

storing an index to said seed value, an offset and said code word in a second format when said number and said image data comprise a number of bits greater than said predetermined number of bits.

2. A method as recited in claim 1, wherein said symbols are JPEG R/S bytes.

3. A method recited in claim 1, wherein said code words are image data.

4. A method as recited in claim 3, wherein said image data is JPEG image data.

5. A method of Huffman decoding compressed data including steps of:

testing bits of a data stream with each of a plurality of test criteria to determine a length of a valid Huffman code,

combining one of a plurality of offsets corresponding to said length with said valid Huffman code to form an index, and

accessing a symbol value in a Huffman table using said index.

6. A method as recited in claim 5, including the further step of:

computing said test criteria and said plurality of offsets from Huffman table data.

7. A method recited in claim 5, wherein said compressed data are image data.

8. A method as recited in claim 7, wherein said image data is JPEG image data.

* * * * *